

Simulation Intelligence: Un paso hacia la Computación Científica

Julián María Galindo Álvarez

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática
Máster Universitario en Lógica, Computación e Inteligencia Artificial

Tutorizado y dirigido por

Dr. Fernando Sancho Caparrini

10 de diciembre de 2022

1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

6 Anexo

1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

6 Anexo

Contexto actual

Modelado y simulación

Implementación de representaciones físicas, matemáticas o lógicas de un sistema, generando datos como base para la toma de decisiones, el análisis, la experimentación y la comprensión del comportamiento del sistema sin necesidad de probarlo en el mundo real.

Coste computacional,
complejidad, sesgos → simplificaciones → simuladores poco
prácticos

Los avances en IA y ML han permitido:

- Introducir conocimientos del dominio
- Maximizar utilidad de los datos
- Generar datos sintéticos
- Cuantificar incertidumbre

pero → reciente y
heterogéneo



1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

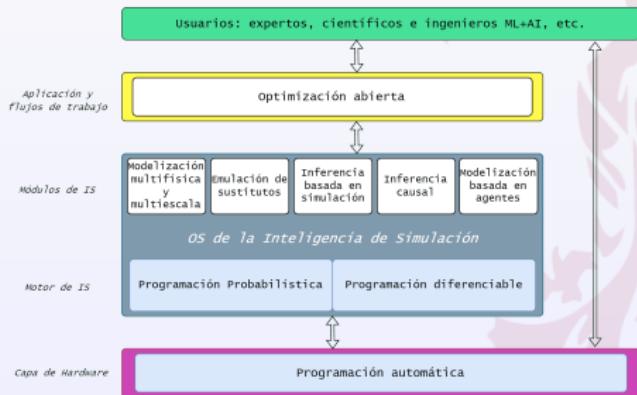
6 Anexo

Simulation Intelligence y Pila SI

Simulation Intelligence

Metodologías necesarias, y cómo deben combinarse de manera efectiva, para lograr avances significativos en simulación e AI en el ámbito científico.

- Autores de múltiples campos
- Perspectiva **unificadora e integradora**
- Fusiona la computación y simulación científica e IA
- Nueve **motivos** principales
- Razonamiento a alto nivel
- Motivos interdependientes

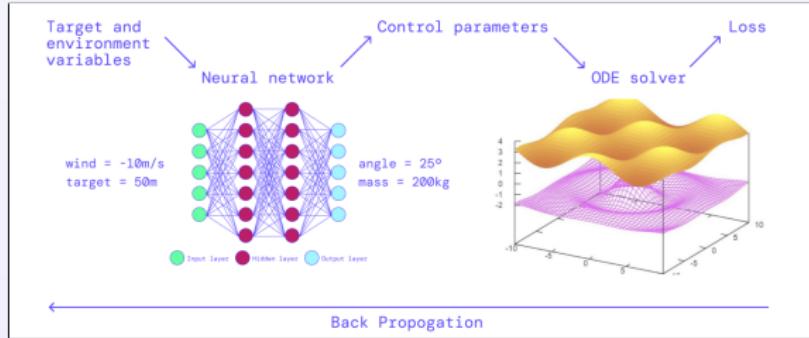


Programación diferenciable

Paradigma PD

Las derivadas de un programa se calculan **automáticamente** y se utilizan en la *optimización por gradiente* de una función de pérdida con el fin de ajustar los parámetros del programa para lograr un objetivo determinado.

- Generalización de *Deep Learning*
- **Diferenciación automática:** interpretación no estándar de un programa

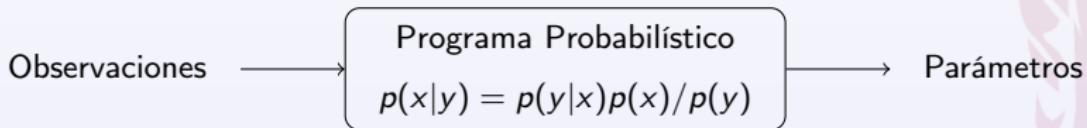


Programación probabilística

Paradigma PP

Equipara los modelos probabilísticos generativos con los programas ejecutables relacionando causas no observables con datos observables para simular cómo se generan los datos en el mundo real.

Lenguajes de Programación Probabilística: se especifican los modelos, y la inferencia se realiza automáticamente.



- Lenguajes de alto nivel
- Comparación de modelos y experimentos de forma rápida
- Permite incorporar conocimiento del dominio

Módulos

● Modelización multifísica y multiescala

- Modelos basados en datos → ignoran leyes físicas
- Necesidad de sesgos *observacionales, inductivos y de aprendizaje*



Módulos

- **Modelización multifísica y multiescala**
 - Modelos basados en datos → ignoran leyes físicas
 - Necesidad de sesgos *observacionales, inductivos y de aprendizaje*
- **Inferencia basada en simulación**
 - Simulaciones estocásticas numéricas
 - Modelos de contraste de hipótesis en base a pruebas empíricas



Módulos

- **Modelización multifísica y multiescala**
 - Modelos basados en datos → ignoran leyes físicas
 - Necesidad de sesgos *observacionales, inductivos y de aprendizaje*
- **Inferencia basada en simulación**
 - Simulaciones estocásticas numéricas
 - Modelos de contraste de hipótesis en base a pruebas empíricas
- **Emulación de sustitutos**
 - Ecuaciones Diferenciales Universales → Aproximadores Universales

Módulos

- **Modelización multifísica y multiescala**
 - Modelos basados en datos → ignoran leyes físicas
 - Necesidad de sesgos *observacionales, inductivos y de aprendizaje*
- **Inferencia basada en simulación**
 - Simulaciones estocásticas numéricas
 - Modelos de contraste de hipótesis en base a pruebas empíricas
- **Emulación de sustitutos**
 - Ecuaciones Diferenciales Universales → Aproximadores Universales
- **Modelización basada en agentes**
 - Teoría de Juegos, emergencia, RL,...
 - Juegos de Markov Multiagente Parcialmente Observables.

Módulos

- **Modelización multifísica y multiescala**
 - Modelos basados en datos → ignoran leyes físicas
 - Necesidad de sesgos *observacionales, inductivos y de aprendizaje*
- **Inferencia basada en simulación**
 - Simulaciones estocásticas numéricas
 - Modelos de contraste de hipótesis en base a pruebas empíricas
- **Emulación de sustitutos**
 - Ecuaciones Diferenciales Universales → Aproximadores Universales
- **Modelización basada en agentes**
 - Teoría de Juegos, emergencia, RL,...
 - Juegos de Markov Multiagente Parcialmente Observables.
- **Inferencia causal**
 - Descubrimiento causal activo

Vanguardia

Optimización abierta:

- *Algoritmo que puede producir sistemáticamente elementos nuevos o mayor complejidad en sus resultados*
- **Inteligencia artificial de desarrollo:** dotar a agentes con sentido de "curiosidad".
- Resolver los problemas de optimización planteados por los profesionales y plantear nuevas preguntas y enfoques.

Programación automática:

- *Automatización del desarrollo de software y potencialmente de hardware*
- Dos enfoques:
 - Estadístico
 - Basado en reglas

1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

6 Anexo



Programación diferenciable y sistema δP

Programación clásica	Programación diferenciable
Secuencia de instrucciones explícitas	Secuencia de primitivas diferenciables
Arquitectura fija	Arquitectura optimizable que busca en un subconjunto de programas posibles
Programas definidos por el usuario	Programas definidos por los datos
Programación imperativa	Programación declarativa, que especifica los objetivos pero no cómo alcanzarlos
Directa, intuitiva y explicable	Alto nivel de abstracción

Un sistema δP debe tener:

- ① Una baja sobrecarga independiente del tamaño de la operación ejecutada.
- ② Soporte completo y eficiente para el flujo de control y tipos de datos definidos por el usuario.
- ③ Personalización.
- ④ Sinergia con código existente ajeno a δP .

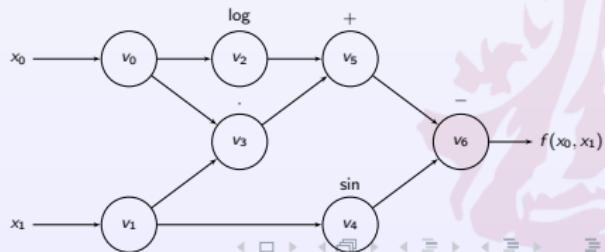
Programa diferenciable

Programa diferenciable: grafo acíclico dirigido (n, m, l, V, E, F)

- ① n y m : dimensiones de entrada y de salida respectivamente.
- ② $N = n + l + m$: número de nodos del grafo y $V = \{v_i\}_{i=0}^{i=N-1}$ es un conjunto de tensores (*variables/nodos*).
 - v_0, \dots, v_{n-1} : *parámetros de entrada*.
 - v_n, \dots, v_{n+l-1} : *variables intermedias*.
 - v_{n+l}, \dots, v_{N-1} : *variables de salida*.
- ③ $E = \{(i, j) : i < j\}$: conjunto de aristas del grafo.
- ④ $F = \{f_i\}_{i=n}^{i=N}$: conjunto de funciones continuamente diferenciables.

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$(x_0, x_1) \mapsto \log(x_0) + x_0 x_1 - \sin(x_1)$$



Diferenciación Automática

Cálculo automático de derivadas evaluadas en base a la regla de la cadena.

Hacia atrás:

Hacia adelante:

- Asocia a cada v_i : $\dot{v}_i = \partial v_i / \partial x_i$
- Inicialización: $\dot{x}_i = \delta_{ij}$ ($\dot{x} = e_i$)
- Una fase: calcula $J_f|_{x=x^*}[i]$
- Eficiente para $f : \mathbb{R} \rightarrow \mathbb{R}^m$
- n iteraciones para gradiente de $f : \mathbb{R}^n \rightarrow \mathbb{R}$

- Asocia a cada v_i : $a_i = \partial y / \partial v_i$
- Inicialización: $a_i = 1$
- Fase hacia adelante
- Fase hacia atrás: $J_f^T|_{x=x^*}$
- Eficiente para $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ con $n \gg m$
- Una iteración para gradiente de $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Inicializando adjunto como r en modo hacia atrás:

$$J_f^T(x^*)r = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \Big|_{x=x^*} \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix}$$

δP y Zygote.jl

Operador Diferencial \mathcal{J}

$$\mathcal{J}(f) := x \mapsto (f(x), z \mapsto J_f^T(x) \cdot z)$$

Si $f : \mathbb{R}^n \rightarrow \mathbb{R}$ entonces $\nabla f = x \mapsto [\mathcal{J}(f)(x)]_2(1)$

```
function J(f ∘ g)(x)
    gx, dg = J(g)(x)
    fgx, dfg = J(f)(gx)
    fgx, z -> dg(dfg(z))
end
```

Estrategia de implementación:

- ① Implementar \mathcal{J} en funciones fundamentales
- ② Interfaz extensible: $\partial(f)(.) = \mathcal{J}(f)(.)$

Ventajas:

- No requiere conocimientos de nuevos tipos
- Gradiéntes personalizados



Ejemplos. Funciones básicas

```
using Zygote
# Gradiente de  $3x^2 + 2x + 1$  en  $x = 5$ 
gradient(x -> 3x^2 + 2x + 1, 5) # (32.0,)

# Gradiente de  $f(x,y) = xy$  en (2,3)
f(x,y) = x*y
gradient(f, [2, 3]) # ([3.0, 2.0],)

# Jacobiano de  $f(x,y) = [x^2, x*y+1]$  en (2,3)
jacobian((x,y) -> [x^2,x*y+1], 2, 3) # ([4, 3], [0, 2])

# Gradiente y Hessiano de  $f(x,y) = xy$  en (2,3)
f(x) = x[1]*x[2]
gradient(f, 2, 3) # (3.0, 2.0)
hessian(f,[2,3])
# 2x2 Matrix{Int64}:
# 0 1
# 1 0
```

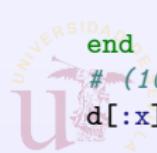


Ejemplos. Estructuras

```
# Definición de potencia recursivamente con bucle for
function pow(x, n)
    r = 1
    for i = 1:n r *= x end
    return r
end
gradient(x -> pow(x, 3), 5) # (75.0,)

# Definición de potencia recursivamente con if (op. ternario)
pow2(x, n) = n <= 0 ? 1 : x*pow2(x, n-1)
gradient(x -> pow2(x, 3), 5) # (75.0,)

# En diccionarios
d = Dict()                      # instancia de diccionario vacío
gradient(5) do x                 # Equivalente a gradient(x -> ..., 5)
    d[:x] = x                     # asigna a la clave :x el valor x
    d[:x] * d[:x]                # devuelve el valor de la clave por sí mismo
end
# (10.0,)
d[:x] # 5 (El diccionario ha sido actualizado)
```



1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

6 Anexo

Neural Ordinary Differential Equations

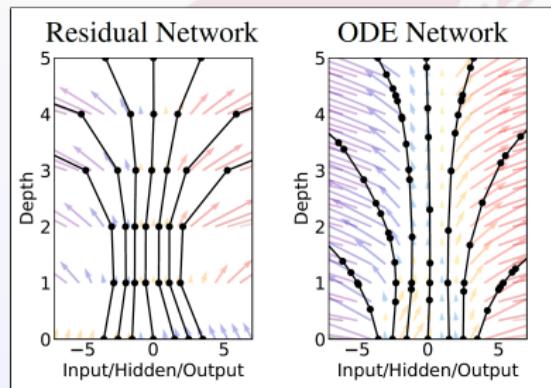
Redes recurrentes:

$$\begin{cases} h_{t+1} = h_t + f(h_t, \theta_t) \\ h_{t_0} = h_0 \\ h_t \in \mathbb{R}^n \quad \forall t \in \{t_0, \dots, t_1\} \end{cases}$$



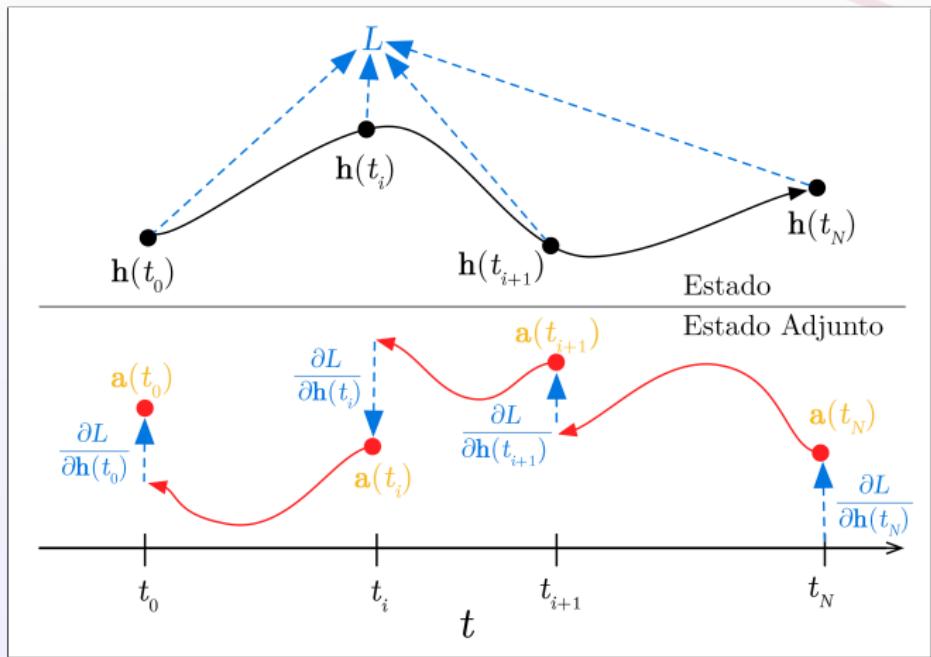
Neural ODEs:

$$\begin{cases} \frac{dh(t)}{dt} = f(h(t), t, \theta) \\ h(t_0) = h_0 \\ h : [t_0, t_1] \rightarrow \mathbb{R}^n \end{cases}$$



Solvers de ODEs de caja negra: → $h(t_1) = \text{ODESolve}(h(t_0, f, t_0, t_1, \theta))$

AD en NODEs



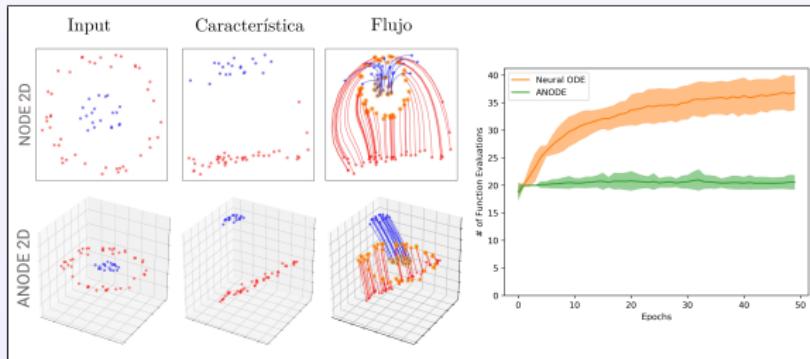
Limitaciones

- ① **Minibatching:** Evaluaciones en los minibatches a través del solver concatenando los estados de cada elemento del batch, creando una ODE combinada de dimensión mayor.
- ② **Tolerancias:** Requiere que el usuario elija una tolerancia de error en los pasos hacia adelante y hacia atrás durante el entrenamiento.
- ③ **Reconstrucción de la trayectoria primaria:** Puede introducir un error numérico adicional si la trayectoria reconstruida diverge de la original.
- ④ **Dinámicas de orden superior:** Sólo contempla dinámicas de primer orden.
- ⑤ **Unicidad:** El *Teorema de Picard* afirma que la solución de un problema de valor inicial existe y es única si la ecuación diferencial es uniformemente continua Lipschitz.

Augmented NODEs

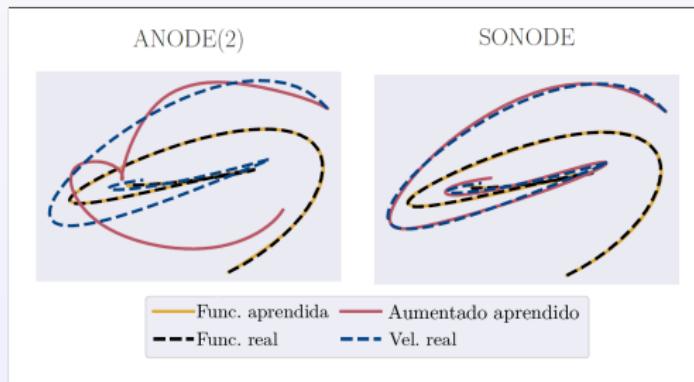
Aumentar p dimensiones para evitar intersecciones: $\alpha : [t_0, t_1] \rightarrow \mathbb{R}^p$ \rightarrow

$$\begin{cases} \frac{d}{dt} \begin{bmatrix} h(t) \\ \alpha(t) \end{bmatrix} = f \left(\begin{bmatrix} h(t) \\ \alpha(t) \end{bmatrix}, t, \theta \right) \\ \begin{bmatrix} h(t_0) \\ \alpha(t_0) \end{bmatrix} = \begin{bmatrix} h_0 \\ 0 \end{bmatrix} \end{cases}$$

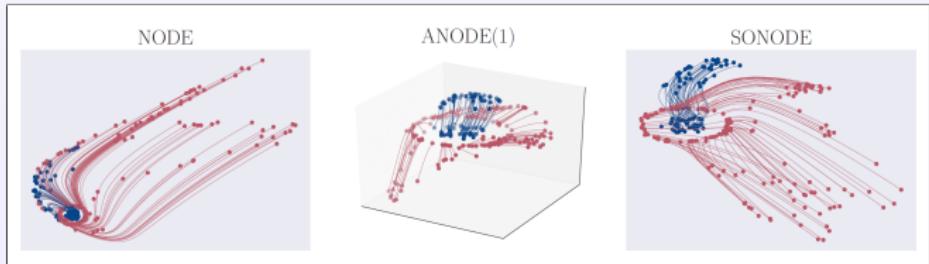
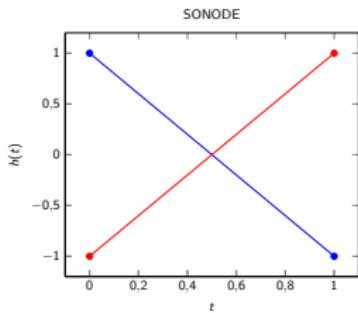
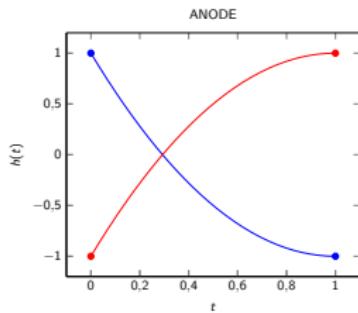
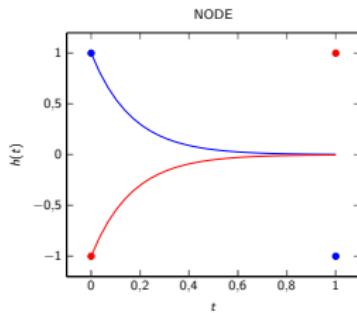


Second Order NODEs

$$\begin{cases} \frac{d^2 h}{dt^2} = f(h, dh/dt, t, \theta_f) \\ h(t_0) = h_0 \\ \left. \frac{dh}{dt} \right|_{t_0} = g(h_0, \theta_g) \end{cases} \longleftrightarrow \begin{cases} \frac{d}{dt} \begin{bmatrix} h(t) \\ \alpha(t) \end{bmatrix} = \begin{bmatrix} \alpha(t) \\ f(h, \alpha, t, \theta_f) \end{bmatrix} \\ \begin{bmatrix} h(t_0) \\ \alpha(t_0) \end{bmatrix} = \begin{bmatrix} h_0 \\ g(h_0, \theta_g) \end{bmatrix} \end{cases}$$



Comparación



DiffEqFlux

DiffEqFlux

Integración de los solvers nativos de ecuaciones diferenciales de *DifferentialEquations* en el paquete de aprendizaje profundo *Flux*.

- Uso del conjunto de métodos de resolución de ecuaciones diferenciales dentro de modelos de ML.
- Marco de desarrollo de modelos NDE y sus variantes.

```
Chain(  
    Dense(28^2, 4, relu),  
    p -> solve(prob,  
        Tsit5(),  
        p=p,  
        saveat=0.1)[1,:],  
    Dense(6, 10),  
    softmax)
```

```
Chain(  
    Conv((2,2), 1=>16, relu),  
    x -> maxpool(x, (2,2)),  
    Conv((2,2), 16=>8, relu),  
    x -> maxpool(x, (2,2)),  
    x -> reshape(x,: ,size(x, 4)),  
    x -> solve(prob,  
        Tsit5(),  
        u0=x,  
        saveat=0.1)[1,:],  
    Dense(288, 10),  
    softmax)
```

```
Chain(  
    Conv((2,2), 1=>16, relu),  
    x -> maxpool(x, (2,2)),  
    Conv((2,2), 16=>8, relu),  
    x -> maxpool(x, (2,2)),  
    x -> reshape(x,: ,size(x, 4)),  
    x -> model_node(x)[1,:],  
    Dense(288, 10),  
    softmax)
```

Ajuste 2D NODE

[Animaciones en Github](#)

[Código](#)



Ajuste 2D ANODE

[Animaciones en Github](#)

[Código](#)



Ajuste cos NODE

[Animaciones en Github](#)

[Código](#)



Ajuste cos ANODE

[Animaciones en Github](#)

[Código](#)



1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

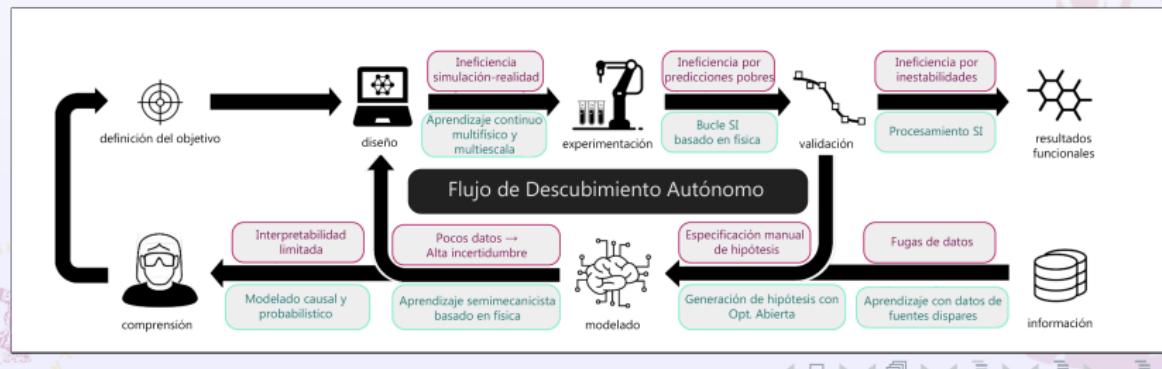
5 Conclusiones

6 Anexo



Conclusiones

- **Simulation Intelligence:** Fusión de la AI, simulación y computación a gran escala en busca de una nueva metodología científica.
- La **Programación Diferenciable** permite el cálculo de derivadas en programas que hasta ahora era difícil de considerar.
- Paradigma de **Neural Differential Equations:** No aproximar o aprender una función sino la dinámica que subyace a la función que pretende aproximarse.



1 Simulación e Inteligencia artificial

2 Simulation Intelligence

- Motor
- Módulos
- Vanguardia

3 Programación Diferenciable

- Diferenciación Automática
- Zygote.jl

4 Neural Differential Equations

- Variantes
- DiffEqFlux

5 Conclusiones

6 Anexo



Ejemplo

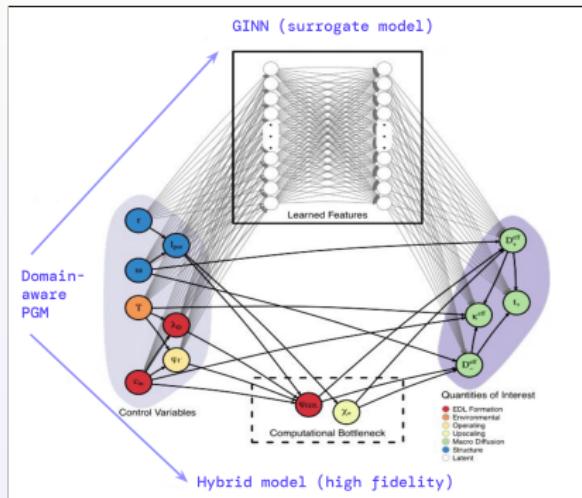


Figura. En la parte inferior de la imagen se representan las variables que representan el cuello de botella, mientras que en la parte superior se representa una red sustituta que reemplaza dichas variables.

Turing.jl: Lotka-Volterra

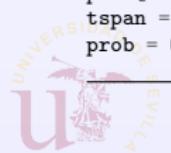
Definición del problema Lotka-Volterra:

```
# Define Lotka-Volterra model.
function lotka_volterra(du, u, p, t)
    # Model parameters.
    α, β, γ, δ = p
    # Current state.
    x, y = u

    # Evaluate differential equations.
    du[1] = (α - β * y) * x # prey
    du[2] = (δ * x - γ) * y # predator

    return nothing
end

# Define initial-value problem.
u0 = [1.0, 1.0]
p = [1.5, 1.0, 3.0, 1.0]
tspan = (0.0, 10.0)
prob = ODEProblem(lotka_volterra, u0, tspan, p)
```



Turing.jl: Lotka-Volterra

Modelo probabilístico:

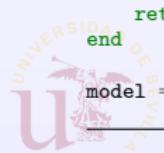
```
using Turing
@model function fitlv(data, prob)
    # Prior distributions.
    α ~ InverseGamma(2, 3)
    β ~ truncated(Normal(1.5, 0.5); lower=0.5, upper=2.5)
    γ ~ truncated(Normal(1.2, 0.5); lower=0, upper=2)
    δ ~ truncated(Normal(3.0, 0.5); lower=1, upper=4)
    δ ~ truncated(Normal(1.0, 0.5); lower=0, upper=2)

    # Simulate Lotka-Volterra model.
    p = [α, β, γ, δ]
    predicted = solve(prob, Tsit5(); p=p, saveat=0.1)

    # Observations.
    for i in 1:length(predicted)
        data[:, i] ~ MvNormal(predicted[i], σ^2 * I)
    end

    return nothing
end

model = fitlv(odedata, prob)
```



Turing.jl: Lotka-Volterra

Resultados de los parámetros estimados y su incertidumbre:

Summary Statistics

parameters	mean	std	naive_se	mcse	ess	rhat
Symbol	Float64	Float64	Float64	Float64	Float64	Float64
α	0.8120	0.0411	0.0008	0.0011	1721.7324	1.0008
α	1.5540	0.0538	0.0010	0.0020	792.3794	1.0012
β	1.0903	0.0534	0.0010	0.0018	913.9769	1.0013
γ	2.8852	0.1426	0.0026	0.0053	815.5472	1.0010
δ	0.9404	0.0506	0.0009	0.0019	804.6753	1.0011